**ESERCIZIO M-1** **–Rilocazione statica con caricamento in partizioni fisse**

In un sistema operativo che utilizza la rilocazione statica e gestisce la memoria con partizioni fisse, la memoria fisica ha un’ampiezza di 10 Mbyte ed è configurata con le seguenti partizioni:

* Partizione 1, ampiezza 1 Mbyte, riservata al Sistema Operativo;
* Partizione 2, ampiezza 0,5 Mbyte, disponibile per processi di ampiezza a con a≤0,5 Mbyte;
* Partizione 3, ampiezza 1 Mbyte, disponibile per processi di ampiezza a con 0,5<a≤1 Mbyte;
* Partizione 4, ampiezza 2,5 Mbyte, disponibile per processi di ampiezza a con 1<a≤2,5 Mbyte;
* Partizione 5, ampiezza 5 Mbyte, disponibile per processi di ampiezza a con 2,5<a≤5 Mbyte.

A partire dal tempo t=0 vengono generati i seguenti processi:

* t= 0 : Processo P1, che occupa 0,3 Mbyte con tempo di completamento di 3 sec;
* t= 0+  : Processo P2, che occupa 0,9 Mbyte con tempo di completamento di 2 sec;
* t= 0+ 2 : Processo P3, che occupa 4 Mbyte con tempo di completamento di 5 sec;
* t= 0+ 3 : Processo P4, che occupa 0,4 Mbyte con tempo di completamento di 1 sec;
* t= 0+ 4 : Processo P5, che occupa 2 Mbyte con tempo di completamento di 4 sec;
* t= 0+ 5 : Processo P6, che occupa 0,7 Mbyte con tempo di completamento di 3 sec;
* t= 0+ 6 : Processo P7, che occupa 3 Mbyte con tempo di completamento di 6 sec;
* t= 0+ 7 : Processo P8, che occupa 0,2 Mbyte con tempo di completamento di 5 sec;
* t= 0+ 8 : Processo P9, che occupa 1,9 Mbyte con tempo di completamento di 2 sec;
* t= 0+ 9 : Processo P10, che occupa 1,2 Mbyte con tempo di completamento di 4 sec;

Dove  è trascurabile. Il caricamento dei processi nelle rispettive partizioni avviene con politica FIFO. Una volta caricato in memoria, ogni processo mantiene l’assegnazione fino alla sua terminazione.

Il processore viene gestito con politica FIFO. Si suppone che, una volta caricati, i processi avanzino fino ala terminazione senza mai sospendersi.

Si chiede a quale tempo avvengono il caricamento in memoria e l’uscita dal sistema di ogni processo.

**SOLUZIONE**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| PROC | LUNGH | CARICATO  al tempo t= | NELLA  PARTIZIONE | IN ESECUZIONE  al tempo t= | TERMINA  al tempo t= |
| P1 | 0,3 | 0 | Part 2 | 0 | 3 |
| P2 | 0,9 | 0+  | Part 3 | 3 | 5 |
| P3 | 4 | 0+ 2 | Part 5 | 5 | 10 |
| P4 | 0,4 | 3 | Part 2 | 14 | 15 |
| P5 | 2 | 0+ 3 | Part 4 | 10 | 14 |
| P6 | 0,7 | 5 | Part 3 | 15 | 18 |
| P7 | 3 | 10 | Part 5 | 18 | 24 |
| P8 | 0,2 | 15 | Part 2 | 26 | 31 |
| P9 | 1,9 | 14 | Part 4 | 24 | 26 |
| P10 | 1,2 | 26 | Part 4 | 31 | 35 |

**ESERCIZIO M-2 Rilocazione statica con caricamento in partizioni fisse, tasso di frammentazione**

In un sistema operativo che utilizza la rilocazione statica e gestisce la memoria con partizioni fisse, la memoria fisica ha un’ampiezza di 10 Mbyte ed è configurata con le seguenti partizioni:

* Partizione 1, ampiezza 1 Mbyte, riservata al Sistema Operativo;
* Partizione 2, ampiezza 0,5 Mbyte, disponibile per processi di ampiezza a con a≤0,5 Mbyte;
* Partizione 3, ampiezza 1 Mbyte, disponibile per processi di ampiezza a con 0,5<a≤1 Mbyte;
* Partizione 4, ampiezza 2,5 Mbyte, disponibile per processi di ampiezza a con 1<a≤2,5 Mbyte;
* Partizione 5, ampiezza 5 Mbyte, disponibile per processi di ampiezza a con 2,5<a≤5 Mbyte.

A un certo sono caricati in memoria i seguenti processi:

* Processo P1, che occupa 0,3 Mbyte;
* Processo P2, che occupa 0,9 Mbyte
* Processo P3, che occupa 2 Mbyte:
* Processo P4, che occupa 3 Mbyte

Si chiede di valutare il tasso di frammentazione della memoria, definito come rapporto tra la somma delle lunghezze dei frammenti inutilizzati e l’ampiezza complessiva della memoria riservata ai processi, e di indicare inizio e fine di ciascun frammento.

**SOLUZIONE**

(Inizio e fine riferiti a inizio partizione)

* Frammento Part 2 inizio 0,3 fine 0,5
* Frammento Part 3 inizio 0,9 fine 1
* Frammento Part 4 inizio 2 fine 2,5
* Frammento Part 5 inizio 3 fine 5

Lunghezza complessiva dei frammenti (Mbyte) 0,2 + 0,1 + 0, 5 + 2 = 2,8

Tasso di frammentazione della memoria : 2,8 / 9

# ESERCIZIO M-3 Rilocazione dinamica e caricamento con partizioni variabili

In un sistema che gestisce la memoria con rilocazione dinamica e caricamento in partizioni variabili, il sistema operativo occupa una partizione con origine 0 e lunghezza 7. Inoltre prima del tempo t=10 sono stati caricati in memoria i seguenti processi:

* Al tempo 1 il processo C, che occupa una partizione con origine 23 e lunghezza 5;
* Al tempo 5 il processo B, che occupa una partizione con origine 16 e lunghezza 7;
* Al tempo 7 il processo A, che occupa una partizione con origine 7 e lunghezza 5;
* Al tempo 9 il processo D, che occupa una partizione con origine 30 e lunghezza 2.

Prima del tempo 10 sono generati nell’ordine anche i seguenti processi, che rimangono in attesa di caricamento:

* il processo E che richiede una partizione di lunghezza 10;
* il processo F che richiede una partizione di lunghezza 5;
* il processo G che richiede una partizione di lunghezza 4.

I processi in attesa di caricamento sono inseriti in una coda *FIFO*. Tutti i tempi sono espressi in *msec* e tutte le lunghezze delle partizioni sono espresse in *Mbyte*.

A ogni processo caricato in memoria viene revocata l’assegnazione dopo che sono trascorsi 10 *msec* dal caricamento. Quando un processo viene scaricato è inserito nell’ultima posizione della coda *FIFO* e il sistema operativo carica (se possibile) uno o piu’ dei processi in attesa di caricamento, adottando la politica FIFO per la scelta dei processi da caricare e la politica *First-Fit* per l’assegnazione delle partizioni*.* Utilizzando il grafico sotto riportato, mostrare come evolvono l’occupazione della memoria e la coda FIFO dei processi in attesa di caricamento a seguito dei primi 6 interventi del sistema operativo, a partire da quello che avviene al tempo 11. Durante questo periodo nessuno dei processi termina.

E

****

t= 9: coda E -> F -> G; t=11: scaricato C t=15: scaricato B t= 17: scaricato A

caricato -- caricati E, F caricato G

coda E -> F -> G -> C coda G -> C -> B C -> B -> A

t= 19 : scaricato D t= 25: scaricati E, F t= 27: scaricato G

caricato C caricati B, A, D caricato --

coda B -> A -> D coda E -> F coda E -> F -> G

# ESERCIZIO M-4 Rilocazione dinamica e caricamento in partizioni variabili

In un sistema che gestisce la memoria con rilocazione dinamica e caricamento in partizioni variabili, il sistema operativo occupa una partizione con origine 0 e lunghezza 7. Prima del tempo 0 sono stati generati e caricati in memoria i seguenti processi

* il processo C, che risiede in memoria da 7 *msec* ed occupa una partizione con origine 23 e lunghezza 5;
* il processo A, che risiede in memoria da 6 *msec* ed occupa una partizione con origine 7 e lunghezza 5;
* il processo D, che risiede in memoria da 5 *msec* ed occupa una partizione con origine 30 e lunghezza 2.
* il processo B, che risiede in memoria da 2 *msec* ed occupa una partizione con origine 16 e lunghezza 7;

Inoltre ai tempi 0, 1 e 2 sono generati rispettivamente i seguenti processi, che fino al tempo 3 rimangono in attesa di caricamento:

* il processo E che richiede una partizione di lunghezza 10;
* il processo F che richiede una partizione di lunghezza 5;
* il processo G che richiede una partizione di lunghezza 6.

Il codice dei processi è rilocabile. I processi in attesa di caricamento sono inseriti in una coda FIFO. Tutti i tempi sono espressi in *msec* e tutte le lunghezze delle partizioni sono espresse in *Mbyte*.

A ogni processo caricato in memoria viene revocata l’assegnazione quando sono trascorsi 10 *msec* dal caricamento. In questa circostanza interviene lo scheduler, che inserisce il processo scaricato nell’ultima posizione della coda dei processi in attesa di caricamento e, finchè il primo processo di questa coda trova una partizione libera di ampiezza sufficiente, lo rimuove dalla coda e lo carica in memoria. Se nessuno dei processi in coda può essere caricato con questa politica, si esegue il compattamento della memoria (verso il basso) e si ripete il tentativo di caricamento La politica per l’assegnazione delle partizioni è la *First Fit.*

Utilizzando il grafico sotto riportato, mostrare come evolvono l’occupazione della memoria e la coda dei processi in attesa di caricamento a seguito dei primi 6 interventi del sistema operativo, a partire da quello che avviene al tempo 3.

Si suppone che nessuno dei processi terminii in questo intervallo di tempo.

****

**ESERCIZIO M-5 Rilocazione dinamica e caricamento in partizioni variabili**

In un sistema operativo che gestisce la memoria con rilocazione dinamica e caricamento in partizioni variabili, la memoria fisica ha un’ampiezza di 35 Mbyte. La partizione 1, della lunghezza di 7 Mbyte, è riservata al sistema operativo, mentre il resto della memoria è disponibile per il caricamento dei processi.

Al tempo *t*  sono caricati in memoria i seguenti processi:

* Processo A, nella partizione con origine 7 Mbyte e lunghezza 5 Mbyte;
* Processo B, nella partizione con origine 16 Mbyte e lunghezza 7 Mbyte;
* Processo C, nella partizione con origine 23 Mbyte e lunghezza 5 Mbyte;
* Processo D, nella partizione con origine 30 Mbyte e lunghezza 2 Mbyte;

Successivamente sii verificano i seguenti eventi (in alternativa):

1. Al tempo *t+2* termina il processo B; quindi al tempo *t+ 5*  viene generato il processo E il cui spazio virtuale occupa 10 Mbyte;
2. Al tempo *t+2* termina il processo C; quindi al tempo *t+ 5*  viene generato il processo F il cui spazio virtuale occupa 5 Mbyte;
3. Al tempo *t+2* viene generato il processo G il cui spazio virtuale occupa 4 Mbyte;quindi al tempo *t+5* termina il processo A.

Per l’assegnazione delle partizioni si utilizza la politica *first fit.*

Per individuare le partizioni libere si utilizza una lista, i cui elementi sono coppie del tipo *(Lunghezza della partizione, Origine della partizione successiva)*

Si chiede, nelle tre ipotesi, la configurazione della lista delle partizioni libere al tempo *t,* al tempo *t+2* e al tempo *t+5,*

**SOLUZIONE**

Al tempo *t*:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SO | SO | SO | SO | SO | SO | SO | A | A | A | A | A |  |  |  |  | B | B | B | B | B | B | B | C | C | C | C | C |  |  | D | D |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |

* Partizioni libere: (0rigine 12, lungh, 4); (origine 28, lungh 2), (origine 32, lungh 3)
* Lista (*Ø, 12) 🡪 (4, 28) 🡪 (2, 32) 🡪 (3, Ø)*

**Ipotesi 1**

**Al tempo *t+2:***

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SO | SO | SO | SO | SO | SO | SO | A | A | A | A | A |  |  |  |  |  |  |  |  |  |  |  | C | C | C | C | C |  |  | D | D |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |

* Partizioni libere: (0rigine 12, lungh, 11); (origine 28, lungh 2), (origine 32, lungh 3)

Lista (*Ø, 12) 🡪 (11, 28 ) 🡪 (2, 32) 🡪 (3, Ø)*

**Al tempo *t+5:***

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SO | SO | SO | SO | SO | SO | SO | A | A | A | A | A | E | E | E | E | E | E | E | E | E | E |  | C | C | C | C | C |  |  | D | D |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |

* Partizioni libere: (origine 22, lungh 1), (origine 28, lungh 2), (origine 32, lungh 3)

Lista (*Ø, 22) 🡪 (1, 28 ) 🡪 (2, 32) 🡪 (3, Ø)*

**ESERCIZIO M-6 Rilocazione dinamica e caricamento in partizioni variabili**

In un sistema operativo che gestisce la memoria con rilocazione dinamica e caricamento in partizioni variabili, la memoria fisica ha un’ampiezza di 35 Mbyte. La partizione 1, della lunghezza di 7 Mbyte, è riservata al sistema operativo, mentre il resto della memoria è disponibile per il caricamento dei processi.

Al tempo *t*  sono caricati in memoria i seguenti processi:

* Processo A, nella partizione con origine 7 Mbyte e lunghezza 5 Mbyte;
* Processo B, nella partizione con origine 16 Mbyte e lunghezza 7 Mbyte;
* Processo C, nella partizione con origine 23 Mbyte e lunghezza 5 Mbyte;
* Processo D, nella partizione con origine 30 Mbyte e lunghezza 2 Mbyte;

Successivamente si verificano i seguenti eventi (in alternativa):

* 1. Al tempo *t+2* termina il processo B; quindi al tempo *t+ 5*  viene generato il processo E il cui spazio virtuale occupa 10 Mbyte;
  2. Al tempo *t+2* termina il processo C; quindi al tempo *t+ 5*  viene generato il processo F il cui spazio virtuale occupa 5 Mbyte;
  3. Al tempo *t+2* viene generato il processo G il cui spazio virtuale occupa 4 Mbyte;quindi al tempo *t+5* termina il processo A.

Per l’assegnazione delle partizioni si utilizza la politica *first fit.*

NOTA: LE PRECDENTI SPECIFICHE SONO LE STESSE DELL’ESERCIZIO M-5

Nelle 3 ipotesi, si chiede di valutare al tempo *t+5* il tasso di frammentazione della memoria, definito come rapporto tra la somma delle lunghezze dei frammenti inutilizzati e l’ampiezza complessiva della memoria riservata ai processi.

**SOLUZIONE**

Al tempo *t:*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SO | SO | SO | SO | SO | SO | SO | A | A | A | A | A |  |  |  |  | B | B | B | B | B | B | B | C | C | C | C | C |  |  | D | D |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |

**Alternativa 1)**

Al tempo *t+2* termina il processo B; quindi al tempo *t+ 5*  viene generato il processo E il cui spazio virtuale occupa 10 Mbyte;

Al tempo *t+2:*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SO | SO | SO | SO | SO | SO | SO | A | A | A | A | A |  |  |  |  |  |  |  |  |  |  |  | C | C | C | C | C |  |  | D | D |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |

Al tempo *t+5:*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SO | SO | SO | SO | SO | SO | SO | A | A | A | A | A | E | E | E | E | E | E | E | E | E | E |  | C | C | C | C | C |  |  | D | D |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |

* Frammento 1 [22, 23)
* Frammento 2 [28, 30)
* Frammento 3 [32, 35)

Lunghezza complessiva dei frammenti (Mbyte): 6

Tasso di frammentazione della memoria (%) 6/28 🡪 = 21,4%

ESERCIZIO M-6, Alternativa 2)

Al tempo *t+2* termina il processo C; quindi al tempo *t+ 5*  viene generato il processo F il cui spazio virtuale occupa 5 Mbyte;

Al tempo *t:*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SO | SO | SO | SO | SO | SO | SO | A | A | A | A | A |  |  |  |  | B | B | B | B | B | B | B | C | C | C | C | C |  |  | D | D |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |

Al tempo *t+2:*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SO | SO | SO | SO | SO | SO | SO | A | A | A | A | A |  |  |  |  | B | B | B | B | B | B | B |  |  |  |  |  |  |  | D | D |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |

Al tempo *t+5:*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SO | SO | SO | SO | SO | SO | SO | A | A | A | A | A |  |  |  |  | B | B | B | B | B | B | B | F | F | F | F | F |  |  | D | D |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |

* Frammento 1 [12, 16)
* Frammento 2 [28, 30)
* Frammento 3 [32, 35)

Lunghezza complessiva dei frammenti (Mbyte): 9

Tasso di frammentazione della memoria 9/28 🡪 = 32,1%

ESERCIZIO M-6 , Alternativa 3)

Al tempo *t+2* viene generato il processo G il cui spazio virtuale occupa 4 Mbyte;quindi al tempo *t+5* termina il processo A.

Al tempo *t:*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SO | SO | SO | SO | SO | SO | SO | A | A | A | A | A |  |  |  |  | B | B | B | B | B | B | B | C | C | C | C | C |  |  | D | D |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |

Al tempo *t+2:*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SO | SO | SO | SO | SO | SO | SO | A | A | A | A | A | G | G | G | G | B | B | B | B | B | B | B | C | C | C | C | C |  |  | D | D |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |

Al tempo *t+5:*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SO | SO | SO | SO | SO | SO | SO |  |  |  |  |  | G | G | G | G | B | B | B | B | B | B | B | C | C | C | C | C |  |  | D | D |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |

* Frammento 1 [7, 12)
* Frammento 2 [28, 30)
* Frammento 3 [32, 35)

Lunghezza complessiva dei frammenti (Mbyte): 10

Tasso di frammentazione della memoria 10/28 🡪 = 36,7%

**ESERCIZIO M-7 Segmentazione**

In un sistema UNIX, la memoria virtuale di ogni processo comprende i segmenti *codice*, *dati* e *pila.* Il caricamento nella memoria fisica avviene con la tecnica delle partizioni variabili.

Il processo ProcK, che occupa le partizioni di origine 21300, 40500 e 62000 e lunghezze 10000, 7000 e 2000 rispettivamente per il codice, i dati e la pila, esegue la chiamata di sistema *fork* che genera il processo ProcJ. I dati e la pila del processo ProcJ sono caricati rispettivamente a partire dagli indirizzi 70000 e 77000.

Si chiedono i contenuti dei registri base e limite dei segmenti codice, dati e pila quando è in esecuzione il processo ProcK e quando è in esecuzione il processo ProcJ.

**SOLUZIONE**

1) Quando è in esecuzione il processo ProcK:

Segmento codice: registro base 21300 registro limite 10000

Segmento dati: registro base 40500 registro limite 7000

Segmento pila: registro base 62000 registro limite 2000

2) Quando è in esecuzione il processo ProcJ:

Segmento codice: registro base 21300 registro limite 10000

Segmento dati: registro base 70000 registro limite 7000

Segmento pila: registro base 77000 registro limite 2000

**ESERCIZIO M-8 Segmentazione**

In un sistema UNIX, la memoria virtuale di ogni processo comprende i segmenti *codice*, *dati* e *pila.* Il caricamento nella memoria fisica avviene con la tecnica delle partizioni variabili.

Il processo ProcK, che occupa le partizioni di origine 21300, 40500 e 62000 e lunghezze 10000, 7000 e 2000 rispettivamente per il codice, i dati e la pila, esegue la chiamata di sistema *fork* che genera il processo ProcJ. Al momento della chiamata sono libere 3 partizioni, che iniziano rispettivamente agli indirizzi 77000, 90000 e 400000 e hanno rispettivamente lunghezze 12000, 10000. e 20000, L’assegnazione delle partizioni avviene con politica *first fit.*

Si chiede:

1. l’origine e la lunghezza delle partizioni che rimangono libere dopo l’esecuzione della *fork ;*
2. i contenuti dei registri base e limite dei segmenti codice, dati e pila quando è in esecuzione il processo ProcK
3. i contenuti dei registri base e limite dei segmenti codice, dati e pila quando è in esecuzione il processo ProcJ.

**SOLUZIONE**

1) Partizioni libere dopo l’esecuzione della fork:

* 1. origine 86000 lunghezza 3000
  2. origine 90000 lunghezza 10000
  3. origine 400000 lunghezza 20000

2) Quando è in esecuzione il processo ProcK:

Segmento codice: registro base 21300 registro limite 10000

Segmento dati: registro base 40500 registro limite 7000

Segmento pila: registro base 62000 registro limite 2000

2) Quando è in esecuzione il processo ProcJ:

Segmento codice: registro base 21300 registro limite 10000

Segmento dati: registro base 77000 registro limite 7000

Segmento pila: registro base 84000 registro limite 2000

**ESERCIZIO M-9 Segmentazione**

In un sistema che gestisce la memoria con spazio logico suddiviso in segmento codice e segmento dati (comprensivo della pila) e caricamento in partizioni variabili con rilocazione dinamica, i registri base e limite del segmento codice hanno a un certo istante i valori B1= 12.000 e L1= 28.000, mentre i registri base e limite del segmento dati hanno i valori B2= 45.000 e L2= 25.000.

Supponendo che il processo in esecuzione estragga istruzioni dagli indirizzi logici 7.000, 30.000 e 25.000, che riferiscono dati con indirizzi logici 47.000, 40.000 e 20.000, dire se ognuno dei precedenti indirizzi logici è legittimo e, in caso affermativo, calcolare il corrispondente indirizzo fisico.

# SOLUZIONE

Istruzioni:

Indirizzo logico Legittimo? Indirizzo fisico

1. 7.000 SI 19000
2. 30.000 NO
3. 25.000 SI 37000

Dati:

1. 47.000 NO
2. 40.000 NO
3. 20.000 SI 65000

**ESERCIZIO M-10 Segmentazione**

In un sistema che gestisce la memoria con segmentazione a domanda, l’indirizzo logico è di 28 bit, dei quali i primi 6 bit codificano l’indice di segmento e i restanti 22 bit definiscono l’offset. Ogni elemento della tabella dei segmenti è formato da 7 byte, dei quali 1 byte contiene indicatori utili al gestore della memoria, 3 byte contengono il valore base, e i restanti 3 byte contengono il valore limite.

Si chiede:

1. Il massimo numero di segmenti di un processo;
2. La massima dimensione di un segmento (in byte);
3. La massima dimensione della tabella dei segmenti (in byte);
4. Il massimo valore possibile per l’indirizzo di origine di un segmento;

**SOLUZIONE**

1. Massimo numero di segmenti di un processo: 26 segmenti = 64 segmenti
2. Massima dimensione di un segmento (in byte): 222 byte = 4 Mbyte
3. Massima dimensione della tabella dei segmenti (in byte): 64\*7= 448 byte
4. Massimo valore possibile per l’indirizzo di origine di un segmento: 224-1

**ESERCIZIO M-11 – Segmentazione in UNIX**

In un sistema simile a Unix che gestisce la memoria con segmentazione e caricamento in partizioni variabili, il sistema operativo occupa una partizione con origine 0 e lunghezza 3. Inoltre al tempo t=10 sono stati caricati in memoria i seguenti processi:

* il processo A, che occupa tre partizioni:
  + - una partizione A1 con origine 9 e lunghezza 2 per il codice
    - una partizione A2 con origine 22 e lunghezza 3 per i dati
    - una partizione A3 con origine 17 e lunghezza 2 per lo stack
* il processo B, che occupa tre partizioni:
  + - una partizione B1 con origine 30 e lunghezza 4 per il codice
    - una partizione B2 con origine 26 e lunghezza 2 per i dati
    - una partizione B3 con origine 3 e lunghezza 2 per lo stack

Tutte le lunghezze delle partizioni sono espresse in *Mbyte* . La memoria ha dimensione 35 e il gestore della memoria adotta politica *best fit* per l’assegnazione delle partizioni.

Al tempo t=11 il processo A esegue una *fork* e viene creato il processo C. Successivamente al tempo 12 il processo C esegue una *exec* per eseguire un nuovo codice. Si supponga che il nuovo codice occupi una partizione C1 di lunghezza 3,e che utilizzi inizialmente una partizione C2 di lunghezza 4 per i dati e una partizione C3 di lunghezza 1 per lo stack.

Utilizzando il grafico sotto riportato, mostrare come evolve l’occupazione della memoria ai tempi 11 e 12.

E

**SOLUZIONE**

****

**t= 11:** Si assegnano al processo C due partizioni, nelle quali si carica una copia dei dati e dello stack del processo padre. Il codice del processo C è condiviso con il padre e pertanto non si deve caricarne una copia.

**t= 12:** Si rilascano le due partizioni precedentemente assegnate al processo C e si assegnano tre nuove partizioni, nelle quali si caricano il codice, i dati e la pila.

**ESERCIZIO M-12 LRU globale**

In un sistema che gestisce la memoria con paginazione, sono presenti i processi A, B e C. Lo stato di occupazione della memoria al tempo 12 è descritto dalla seguente *CoreMap*, dove per ogni blocco si specifica nell’ordine: il processo a cui appartiene la pagina caricata, l’indice della pagina e il tempo al quale è avvenuto l’ultimo riferimento alla pagina stessa . Ad esempio, nel blocco 11 è caricata la pagina 6 del processo B, il cui ultimo riferimento è avvenuto al tempo 5.

SO SO SO SO SO SO B,3 A,1 B,0 C,1 B,6 C,7 A,4 C,3 A,5 C,5 B,2 A,7

11 2 10 3 5 8 12 6 9 7 1 4

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

Per la gestione della memoria si utilizza un algoritmo di sostituzione LRU globale.

Quali pagine vengono scaricate dalla memoria e caricate in memoria ai tempi t= 13, t=14 e t=15 nelle seguenti ipotesi (alternative):

1. Il processo A riferisce la pagina 2 al tempo 13, la pagina 6 al tempo 14 e la pagina 4 al tempo 15;
2. Il processo A riferisce la pagina 5 al tempo 13, la pagina 3 al tempo 14 e la pagina 8 al tempo 15;
3. Il processo B riferisce la pagina 2 al tempo 13, la pagina 7 al tempo 14 e la pagina 6 al tempo 15;
4. Il processo B riferisce la pagina 1 al tempo 13, la pagina 7 al tempo 14 e la pagina 0 al tempo 15;
5. Il processo C riferisce la pagina 0 al tempo 13, la pagina 6 al tempo 14 e la pagina 4 al tempo 15;
6. Il processo C riferisce la pagina 3 al tempo 13, la pagina 4 al tempo 14 e la pagina 0 al tempo 15.

# SOLUZIONE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | t= | Pagina scaricata | Pagina caricata ... | Nel blocco |
| (a) | 13 | - | A2 | 10 |
| 14 | B2 | A6 | 17 |
| 15 | - | - |  |
| (b) | 13 | - | - |  |
| 14 | - | A3 | 10 |
| 15 | B2 | A8 | 17 |
| (c) | 13 | - | - |  |
| 14 | - | B7 | 10 |
| 15 | - | - |  |
| (d) | 13 | - | B1 | 10 |
| 14 | B2 | B7 | 17 |
| 15 | - | - |  |
| (e) | 13 | - | C0 | 10 |
| 14 | B2 | C6 | 17 |
| 15 | A1 | C4 | 7 |
| (f) | 13 | - | - |  |
| 14 | - | C4 | 10 |
| 15 | B2 | C0 | 17 |

**ESERCIZIO M-13 LRU locale**

In un sistema che gestisce la memoria con paginazione, sono presenti i processi A, B e C. Lo stato di occupazione della memoria al tempo 11 è descritto dalla seguente tabella, dove per ogni blocco si specifica nell’ordine: il processo a cui appartiene la pagina caricata, l’indice della pagina e il tempo al quale è avvenuto l’ultimo riferimento alla pagina stessa . Ad esempio, nel blocco 11 è caricata la pagina 6 del processo B, il cui ultimo riferimento è avvenuto al tempo 5.

SO SO SO SO SO SO A,1 B,0 C,1 B,6 C,7 C,3 A,5 C,5 B,2 A,7

2 10 3 5 8 6 9 7 1 4

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

Le tabelle delle pegine dei tre processi sono le seguenti (con notazione evidente; in parentesi è replicato il tempo al quale è avvenuto l’ultimo riferimento)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Pagina | Blocco |  | Pagina | Blocco |  | Pagina | Blocco |
| 0 | - |  | 0 | 8 (10) |  | 0 | - |
| 1 | 7 (2) |  | 1 | - |  | 1 | 9 (3) |
| 2 | - |  | 2 | 17 (1) |  | 2 | - |
| 3 | - |  | 3 | - |  | 3 | 14 (6) |
| 4 | - |  | 4 | - |  | 4 | - |
| 5 | 15 (9) |  | 5 | - |  | 5 | 16 (7) |
| 6 | - |  | 6 | 11 (5) |  | 6 | - |
| 7 | 18 (4) |  | 7 | - |  | 7 | 12 (8) |
| Processo A | |  | Processo B | |  | Processo C | |

Per la gestione della memoria si utilizza un algoritmo di sostituzione LRU locale. Il working set assegnato al processo A (inteso come numero di blocchi a disposizione del processo, anche se momentaneamente non occupati) ha dimensione 3; analogamente quelli assegnati ai processi B e C hanno dimensione 5.

Quali pagine vengono scaricate dalla memoria e caricate in memoria al tempo t= 12 e al tempo 13 nelle seguenti ipotesi (alternative):

1. Il processo A riferisce la pagina 2 al tempo 12 e la pagina 6 al tempo 13;
2. Il processo A riferisce la pagina 5 al tempo 12 e la pagina 3 al tempo 13;
3. Il processo B riferisce la pagina 2 al tempo 12 e la pagina 7 al tempo 13;
4. Il processo B riferisce la pagina 1 al tempo 12 e la pagina 7 al tempo 13;
5. Il processo C riferisce la pagina 0 al tempo 12 e la pagina 6 al tempo 13;
6. Il processo C riferisce la pagina 3 al tempo 12 e la pagina 4 al tempo 13;

# SOLUZIONE

1. t=12: scaricata 1 caricata 2 nel blocco 7 t=13: scaricata 7 caricata 6 nel blocco 18
2. t=12: scaricata - caricata - t=13: scaricata 1 caricata 3 nel blocco 7
3. t=12: scaricata - caricata - t=13: scaricata - caricata 7 nel blocco 6
4. t=12: scaricata - caricata 1 nel blocco 6 t=13: scaricata - caricata 7 nel blocco 10
5. t=12: scaricata - caricata 0 nel blocco 6 t=13: scaricata 1 caricata 6 nel blocco 9
6. t=12: scaricata - caricata - t=13: scaricata - caricata 4 nel blocco 6

**ESERCIZIO M-14 Distanza passata approssimata**

Si consideri un sistema che gestisce la memoria con paginazione a domanda utilizzando un algoritmo LRU. La tabella delle pagine di ogni processo contiene, per ogni pagina, i campi *blocco, P* (bit di presenza), *R* (bit di uso) e *DP* (contatore che registra la distanza passata approssimata della pagina). La distanza passata è aggiornata periodicamente dal processo di sistema *DPM,* che in base al valore del bit *R* azzera o incrementa il contatore *DP.*

Al tempo *t,* quando viene attivato il processo *DPM,* la configurazione della tabella delle pagine del processo *P* è mostrata in figura. Si chiede di completare il contenuto della tabella delle pagine al tempo *t+ k,* quando il processo *DPM* ha terminato l’aggiornamento della distanza passata*.*

**SOLUZIONE**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tabella delle pagine al tempo *t* | | | | |  | Tabella delle pagine al tempo *t+ k* | | | | |
| Pagina | blocco | P | R | DP |  | Pagina | Blocco | P | R | DP |
| 0 | - | 0 | - | - |  | 0 | - | 0 | - | - |
| 1 | 6 | 1 | 0 | 2 |  | 1 | 6 | 1 | 0 | 3 |
| 2 | 7 | 1 | 1 | 3 |  | 2 | 7 | 1 | 0 | 0 |
| 3 | - | 0 | - | - |  | 3 | - | 0 | - | - |
| 4 | 25 | 1 | 0 | 6 |  | 4 | 25 | 1 | 0 | 7 |
| 5 | 8 | 1 | 1 | 5 |  | 5 | 8 | 1 | 0 | 0 |
| 6 | - | 0 | - | - |  | 6 | - | 0 | - | - |
| 7 | 10 | 1 | 0 | 4 |  | 7 | 10 | 1 | 0 | 5 |
| 8 | 15 | 1 | 1 | 8 |  | 8 | 15 | 1 | 0 | 0 |
| 9 | - | 0 | - | - |  | 9 | - | 0 | - | - |
| 10 | 20 | 1 | 0 | 9 |  | 10 | 20 | 1 | 0 | 10 |
| 11 | - | 0 | - | - |  | 11 | - | 0 | - | - |
| 12 | 30 | 1 | 1 | 7 |  | 12 | 30 | 1 | 0 | 0 |

**ESERCIZIO M-15 Algoritmo Second Chance**

In un sistema che gestisce la memoria con paginazione, sono presenti i processi A, B e C. Lo stato di occupazione della memoria è descritto dalla *Core Map*, dove per ogni blocco si specifica nell’ordine: il processo a cui appartiene la pagina caricata, l’indice della pagina caricata e il bit di pagina riferita.

Al tempo *t* la configurazione della *CoreMap* è quella riportata in figura, dove, per esempio, nel blocco 11 è caricata la pagina 7 del processo C con bit di pagina riferita uguale a 1.

L’algoritmo di sostituzione è il *Second Chance* (globale) e prima di ogni invocazione il puntatore è posizionato sul blocco successivo alla *vittima* individuata nell’invocazione precedente.

A,2 A,0 B,3 B,8 C,0 B,7 A,1 B,0 C,1 C,4 B,6 C,7 C,8 A,5 C,3 A,4 C,5 B,2 A,7

1 0 1 1 1 1 1 0 1 0 1 1 1 0 1 1 1 0 1 4

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

Mostrare come si modifica la *CoreMap* se al tempo *t* si verifica (in alternativa) uno dei seguenti eventi:

1. Il puntatore è posizionato sul blocco 1 e il processo A riferisce la pagina 5
2. Il puntatore è posizionato sul blocco 2 e il processo C riferisce la pagina 2
3. Il puntatore è posizionato sul blocco 7 e il processo B riferisce la pagina 5
4. Il puntatore è posizionato sul blocco 18 e il processo A riferisce la pagina 3.

# SOLUZIONE

1) Il puntatore è posizionato sul blocco 1 e il processo A riferisce la pagina 5

Configurazione iniziale:

A,2 A,0 B,3 B,8 C,0 B,7 A,1 B,0 C,1 C,4 B,6 C,7 C,8 A,5 C,3 A,4 C,5 B,2 A,7

1 0 1 1 1 1 1 0 1 0 1 1 1 0 1 1 1 0 1 4

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

Configurazione finale:

A,2 A,0 B,3 B,8 C,0 B,7 A,1 B,0 C,1 C,4 B,6 C,7 C,8 A,5 C,3 A,4 C,5 B,2 A,7

1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 0 1 4

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

2) Il puntatore è posizionato sul blocco 2 e il processo C riferisce la pagina 2

Configurazione iniziale:

A,2 A,0 B,3 B,8 C,0 B,7 A,1 B,0 C,1 C,4 B,6 C,7 C,8 A,5 C,3 A,4 C,5 B,2 A,7

1 0 1 1 1 1 1 0 1 0 1 1 1 0 1 1 1 0 1 4

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

Configurazione finale:

A,2 A,0 B,3 B,8 C,0 B,7 A,1 C,2 C,1 C,4 B,6 C,7 C,8 A,5 C,3 A,4 C,5 B,2 A,7

1 0 0 0 0 0 0 1 1 0 1 1 1 0 1 1 1 0 1 4

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

3) Il puntatore è posizionato sul blocco 2 e il processo C riferisce la pagina 2

Configurazione iniziale:

A,2 A,0 B,3 B,8 C,0 B,7 A,1 B,0 C,1 C,4 B,6 C,7 C,8 A,5 C,3 A,4 C,5 B,2 A,7

1 0 1 1 1 1 1 0 1 0 1 1 1 0 1 1 1 0 1 4

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

Configurazione finale:

A,2 A,0 B,3 B,8 C,0 B,7 A,1 B,5 C,1 C,4 B,6 C,7 C,8 A,5 C,3 A,4 C,5 B,2 A,7

1 0 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 0 1 4

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

4) Il puntatore è posizionato sul blocco 18 e il processo A riferisce la pagina 3.

Configurazione iniziale:

A,2 A,0 B,3 B,8 C,0 B,7 A,1 B,0 C,1 C,4 B,6 C,7 C,8 A,5 C,3 A,4 C,5 B,2 A,7

1 0 1 1 1 1 1 0 1 0 1 1 1 0 1 1 1 0 1 4

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

Configurazione finale:

A,2 A,3 B,3 B,8 C,0 B,7 A,1 B,0 C,1 C,4 B,6 C,7 C,8 A,5 C,3 A,4 C,5 B,2 A,7

0 1 1 1 1 1 1 0 1 0 1 1 1 0 1 1 1 0 0 4

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

**ESERCIZIO M-16 – Tabelle delle pagine**

Si consideri un sistema dove gli indirizzi logici hanno la lunghezza di 32 bit e le pagine logiche e fisiche hanno ampiezza di 4 kByte.Per la gestione della memoria con paginazione dinamica si utilizzano tabelle delle pagine a 2 livelli. La tabella di primo livello comprende 210 elementi.

Gli elementi di ogni tabella di primo o secondo livello occupano 4 byte, di cui 1 è riservato agli indicatori (pagina caricata, riferita, modificata, ecc.) mentre i rimanenti individuano un indice di blocco fisico.

Si chiede:

1. la lunghezza del campo offset, in numero di bit;
2. la lunghezza delle tabelle di secondo livello (numero di elementi);
3. lo spazio occupato in memoria da ogni tabella di secondo livello (numero di byte);
4. la massima dimensione della memoria fisica (numero di blocchi e di byte, espressi come potenze di 2).

# SOLUZIONE

1. lunghezza del campo offset : 12 bit
2. lunghezza (numero di elementi) di ogni tabella di secondo livello: 210 elementi
3. spazio occupato in memoria da ogni tabella di secondo livello (numero di byte) : 212 byte
4. massima dimensione della memoria fisica : 224 blocchi 🡪 224 \* 212 = 236 byte (gli indici dei blocchi di memoria fisica sono codificati con 3 byte)

# ESERCIZIO M- 17 Tabella delle pagine

Un sistema gestisce la memoria con paginazione e usa indirizzi logici di 32 bit, blocchi di memoria di 1 kbyte e tabelle delle pagine a due livelli. La tabella delle pagine di 1° livello (o *directory)* e quelle di 2° livello hanno uguale dimensione. Ogni elemento della tabella di primo livello e di quelle di secondo livello contiene l’indice di un blocco di memoria e 2 bit di controllo.

La memoria fisica ha una capacità di 4 Gbyte (= 232 byte)

Domande:

1. Nell’indirizzo logico, quanti bit sono riservati all’offset nella tabella di 1° livello, quanti all’offset nella tabella di 2° livello e quanti all’offset nella pagina logica?
2. Quanti bit sono necessari per codificare un indice di blocco e quanti bit contiene ogni elemento dei tabella di primo o di secondo livello?
3. Quanti byte e quanti blocchi di memoria occupa ogni tabella?

# SOLUZIONE

Il numero di blocchi fisici è 222 .

1. Offset in Tab 1° livello (numero di bit): 11 bit

Offset in Tab 2° livello (numero di bit): 11 bit

Offset in pagina logica (numero di bit): 10 bit

1. Bit necessari per codificare un indice di blocco: 22 bit

Lunghezza in bit di ogni elemento di tabella: 24 bit (22 bit per l’indice di blocco + 2 bit di controllo)

1. Numero di byte occupati da ogni tabella:

Numero di blocchi occupati da ogni tabella: 211 elementi di 3 byte 🡺 6 \* 210 byte 🡺 6 blocchi

# ESERCIZIO M- 18 Tabella delle pagine

Si consideri un sistema che gestisce la memoria con paginazione dinamica con le seguenti caratteristiche:

* indirizzi logici di 32 bit e ampiezza dello spazio logico di ogni processo pari a 232 byte;
* pagine logiche e blocchi fisici di 1 KByte;
* tabelle delle pagine a due livelli; la tabella di primo livello comprende 210 elementi;
* tutti gli elementi della tabella di primo e di secondo livello hanno lunghezza pari a 32 bit, di cui 10 sono indicatori e i restanti 22 codificano l’indice di un blocco fisico;

In questo sistema è presente il processo P, che ha allocato nello spazio virtuale due aree di memoria:

* + Area 1: 2 Mbyte a partire dalla locazione 100\* 1024
  + Area 2: 4 Mbyte a partire dalla locazione 4096\* 1024

Si chiede:

1. La lunghezza, in numero di bit, delle componenti dell’indirizzo logico che indirizzano, rispettivamente, la tabella di primo livello, di secondo livello, e l’offset;
2. Lo spazio occupato in memoria dalla tabella di primo livello e da ogni tabella di secondo livello (in byte);
3. La massima dimensione della memoria fisica (numero di blocchi e di byte, espressi come potenze di 2);
4. Lo spazio indirizzabile da una tabella di secondo livello;
5. Quali elementi della tabella di primo livello sono usati per indirizzare l’Area 1;
6. Quali elementi della tabella/ delle tabelle di secondo livello sono usate per indirizzare l’Area 1;
7. Quali elementi della tabella di primo livello sono usati per indirizzare l’Area 2;
8. Quali elementi della tabella/ delle tabelle di secondo livello sono usate per indirizzare l’Area 2;

**Soluzione**

1. Lunghezza del campo che indirizza la tabella di 1° livello: 10 bitLunghezza del campo che indirizza la tabella di 2° livello: 12 bit Lunghezza del campo offset: 10 bit
2. Spazio occupato in memoria dalla tabella di 1° livello: 4 kByteSpazio occupato in memoria dalla tabella di 2° livello: 16 kByte
3. Massima dimensione della memoria fisica: 222 blocchi = 222+10 Bytes
4. Spazio indirizzabile da una tabella di 2° livello: 212+10 Bytes = 4 MBytes
5. L’Area 1 occupa le locazioni comprese tra 100\*1024 e (100+2048) \*1024 – 1, quindi tra 100 Kbyte (estremo incluso) e 2Mbyte+100Kbyte (estremo escluso). Si tratta di locazioni comprese nei primi 4Mbyte della memoria virtuale, che sono indirizzabili dalla prima tabella di secondo livello. Quindi l’Area 1 è indirizzata dall’elemento 0 della tabella di primo livello.
6. Ogni elemento di una tabella di 2° livello indirizza 1 Kbyte, quindi l’Area 1 è indirizzata dagli elementi della 1° tabella di secondo livello compresi tra: 100 e 100+2048-1
7. L’Area 2 occupa le locazioni comprese tra 4096\* 1024 e (4096+4096) \*1024 – 1, quindi tra 4 MByte (estremo incluso)e 8MByte (estremo escluso). Si tratta di locazioni comprese nei secondi 4Mbyte della memoria virtuale, che sono indirizzabili dalla seconda tabella di secondo livello. Quindi l’Area 2 è indirizzata dall’elemento 1 della tabella di primo livello.
8. Ogni elemento di una tabella di 2° livello indirizza 1 Kbyte, quindi l’Area 2 è indirizzata dagli elementi della 1° tabella di secondo livello compresi tra 0 e 4096- 1. In altri termini, per indirizzare l’Area 2 si utilizzano tutti gli elementi della tabella di secondo livello di indice 1.

# ESERCIZIO M- 19 Tabella delle pagine

Si consideri un sistema dove gli indirizzi logici hanno la lunghezza di 32 bit e le pagine logiche e i blocchi fisici hanno ampiezza di 2 kByte.Per la gestione della memoria con paginazione a domanda si utilizzano tabelle delle pagine a 3 livelli.

Le tabelle di primo, secondo o terzo livello hanno tutte uguale lunghezza.Per ogni tabella, gli elementi occupano 3 byte e riservano 4 bit agli indicatori (bit di presenza, riferimento, modifica e protezione in scrittura). I rimanenti bit codificano un indice di blocco fisico.

Si chiede:

1. la lunghezza del campo offset, in numero di bit;
2. la lunghezza delle tabelle di secondo livello (numero di elementi);
3. lo spazio occupato in memoria da ogni tabella di secondo livello (numero di byte);
4. la massima estensione della memoria fisica (numero di blocchi e di byte, espressi come potenze di 2).

# SOLUZIONE

1. lunghezza del campo offset : 11 bit
2. lunghezza, in numero di elementi, di ogni tabella di primo, secondo o terzo livello: 27 elementi
3. spazio occupato da ogni tabella di primo, secondo o terzo livello, in numero di byte : 3\* 27 = 384 byte
4. massima estensione della memoria fisica : 220 blocchi 🡪 220 \* 211 = 231 byte 🡪 2 Gbyte.

# ESERCIZIO M- 20 Segmentazione paginata

In un sistema che gestisce la memoria con segmentazione paginata, l’indirizzo logico è di 32 bit, dei quali i primi 6 bit codificano l’indice di segmento e i restanti 26 bit definiscono l’offset all’interno del segmento. I 26 bit dell’offset codificano l’indice di pagina con 14 bit e l’offset all’interno della pagina con i restanti 12 bit.

Si chiede:

* il massimo numero di segmenti di un processo,
* la dimensione di una pagina (in byte);
* la massima dimensione di un segmento (in numero di pagine);
* la massima dimensione della tabella delle pagine (in numero di descrittori).

**SOLUZIONE**

1. Massimo numero di segmenti di un processo: 26 segmenti = 64 segmenti
2. Dimensione di una pagina: 212= 4KB
3. Massima dimensione di un segmento : 214= 16K pagine
4. Massimo numero di descrittori nella tabella delle pagine: 214= 16K

**ESERCIZIO M-21 -Page daemon**

Un sistema operativo simile a UNIX, che gestisce la memoria con paginazione a domanda, utilizza il processo *PageDaemon,* con parametri *lotsfree= 5 e minfree=2,* e l’algoritmo di sostituzione *Second Chance.* Gli elementi della *CoreMap* hanno i campi *Proc* (processo a cui è assegnato il blocco; il campo è vuoto se il blocco è libero); *Pag* (pagina del processo caricata nel blocco), *Rif* (bit di pagina riferita utilizzato da *Second* *Chance*). Al tempo *t* sono presenti i processi A, B, C, D e la *Core Map* ha la configurazione mostrata in figura, con il puntatore dell’algoritmo di sostituzione posizionato sul blocco 11.I primi 6 blocchi della memoria fisica sono riservati al sistema operativo e sono ignorati dall’algoritmo di sostituzione.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Proc |  |  |  |  |  |  | C | A | B | A |  | B | C | D | D |  | D | B | A | B |  | C |  | C |
| Pag |  |  |  |  |  |  | 0 | 1 | 0 | 2 |  | 6 | 3 | 1 | 2 |  | 6 | 2 | 7 | 3 |  | 7 |  | 2 |
| Rif |  |  |  |  |  |  | 0 | 1 | 0 | 0 |  | 1 | 1 | 0 | 1 |  | 0 | 0 | 1 | 0 |  | 1 |  | 0 |
| Blocco | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| Core Map al tempo *t* | | | | | | | | | | | | | | | | | | | | | | | | |

Il *PageDaemon* interviene al tempo *t*  e successivamente ogni 10 msec. Ad ogni intervento, *PageDaemon* avanza per 1 msec occupando in modo esclusivo il processore e scarica fino a 3 pagine o, in alternativa, esegue lo *swapout* di un processo.

La selezione dei processi candidati allo *swapout* avviene in ordine alfabetico. In caso di errori di pagina, i blocchi liberi vengono assegnati in ordine crescente di indice.

Considerare la seguente evoluzione del sistema:

1. Dal tempo *t* al tempo *t+1* avanza il processo *PageDaemon*;
2. Dal tempo *t+1* al tempo *t+10* avanzano i processi B e C, che riferiscono nell’ordine le pagineB0, B1, B6, C2, C0, C4, B2, B5
3. Dal tempo *t+10* al tempo *t+11* avanza il processo *PageDaemon*;
4. Dal tempo *t+11* al tempo *t+20* avanzano i processi D e B, che riferiscono nell’ordine le pagineD3, D6, D2, D1, B0, B2, B3, B0;
5. Dal tempo t+20 al tempo t+21 avanza il processo *PageDaemon.*

Mostrare la configurazione della *CoreMap*  ai tempi 1, 10, 11, 20 e 21

###### SOLUZIONE

(Modificare incrementalmente la configurazione iniziale della *CoreMap*, aggiornando anche la posizione del puntatore)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Proc |  |  |  |  |  |  | C | A | B | A |  | B | C |  | D |  | D | B | A | B |  | C |  | C |
| Pag |  |  |  |  |  |  | 0 | 1 | 0 | 2 |  | 6 | 3 |  | 2 |  | 6 | 2 | 7 | 3 |  | 7 |  | 2 |
| Rif |  |  |  |  |  |  | 0 | 1 | 0 | 0 |  | 0 | 0 |  | 1 |  | 0 | 0 | 1 | 0 |  | 1 |  | 0 |
| Blocco | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| Core Map al tempo *t+1 (scaricata 1 pagina)* | | | | | | | | | | | | | | | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Proc |  |  |  |  |  |  | C | A | B | A | B | B | C | C | D | B | D | B | A | B |  | C |  | C |
| Pag |  |  |  |  |  |  | 0 | 1 | 0 | 2 | 1 | 6 | 3 | 4 | 2 | 5 | 6 | 2 | 7 | 3 |  | 7 |  | 2 |
| Rif |  |  |  |  |  |  | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |  | 1 |  | 1 |
| Blocco | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| Core Map al tempo *t+10 (2 blocchi liberi)* | | | | | | | | | | | | | | | | | | | | | | | | | |
| Proc |  |  |  |  |  |  | C | A | B |  | B | B | C | C | D | B |  | B | A |  |  | C |  | C |
| Pag |  |  |  |  |  |  | 0 | 1 | 0 |  | 1 | 6 | 3 | 4 | 2 | 5 |  | 2 | 7 |  |  | 7 |  | 2 |
| Rif |  |  |  |  |  |  | 0 | 0 | 0 |  | 1 | 1 | 0 | 1 | 0 | 0 |  | 0 | 0 |  |  | 0 |  | 0 |
| Blocco | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| Core Map al tempo *t+11 (scaricate 3 pagine)* | | | | | | | | | | | | | | | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | | | | | | | | | | | | | | |
| Proc |  |  |  |  |  |  | C | A | B | D | B | B | C | C | D | B | D | B | A | D | B | C |  | C |
| Pag |  |  |  |  |  |  | 0 | 1 | 0 | 3 | 1 | 6 | 3 | 4 | 2 | 5 | 6 | 2 | 7 | 1 | 3 | 7 |  | 2 |
| Rif |  |  |  |  |  |  | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |  | 0 |
| Blocco | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

Core Map al tempo *t+20 (1 blocco libero)*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | | | | | | | | | | | | | | |
| Proc |  |  |  |  |  |  | C |  | B | D | B | B | C | C | D | B | D | B |  | D | B | C |  | C |
| Pag |  |  |  |  |  |  | 0 |  | 0 | 3 | 1 | 6 | 3 | 4 | 2 | 5 | 6 | 2 |  | 1 | 3 | 7 |  | 2 |
| Rif |  |  |  |  |  |  | 0 |  | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |  | 1 | 1 | 0 |  | 0 |
| Blocco | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

Core Map al tempo *t+21*

*(scaricato il processo A: il numero di pagine libere rimane < lotsfree, ma PagerDaemon ha il vincolo di scaricare un solo processo in ogni intervento) )*

**ESERCIZIO M-22 -Page daemon**

Un sistema operativo simile a UNIX, che gestisce la memoria con paginazione a domanda, utilizza il processo *PageDaemon,* con parametri *lotsfree= 10 e minfree=4,* e l’algoritmo di sostituzione *Second Chance.* Per l’eventuale *swapout,* i processi sono considerati in ordine alfabetico. Notare che non si considera il parametro *desfree.*

Gli elementi della *CoreMap* hanno i campi *Proc* (processo a cui è assegnato il blocco), *Pag* (pagina del processo caricata nel blocco) e *Rif* (bit di uso, utilizzato da *Second* *Chance*); questi campi sono vuoti se il blocco è libero. I blocchi di indici 0, 1, 2, 3, 4, e 5 sono riservati al sistema operativo e sono ignorati dal *PageDaemon*.

Il *PageDaemon* interviene al tempo *t* sono quando sono presenti i processi A, B, C, D, E e rimane in esecuzione per tutto il tempo necessario ad applicare la sua politica, utilizzando in modo esclusivo il processore (in altri termini, nessuno dei processi avanza in questo intervallo di tempo.

Considerando le ipotesi a), b), c) d) per la configurazione della *CoreMap* e per la posizione del puntatore al tempo *t,* definire per ciascuna ipotesi la configurazione della *CoreMap* e la posizione del puntatore al termine dell’intervento del *PageDaemon* (riempire solo le caselle modificate). Inoltre elencare, con formato (proc, pag), le pagine eventualmente scaricate o i processi eventualmente scaricati.

**SOLUZIONE**

**Ipotesi a)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S. O. | S. O. | S. O. | S. O. | S. O. | S. O. | A  2  1 | A  3  0 | B  2  1 | B  4  1 | A  5  0 | ∅ | C  3  1 | C  5  1 | C  0  0 | D  1  1 | ∅ | ∅ | A  0  1 | ∅ | B  5  0 | B  1  1 | C  7  1 | C  9  1 | C  8  1 | E  0  1 | D  8  1 | A  9  0 | ∅ | A  7  1 | E  1  0 | E  2  1 | ∅ | B  9  1 | C  6  0 | B  11  1 | C  4  0 | ∅ | B  12  1 | E  3  1 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |

Configurazione della *CoreMap* e posizione del puntatore al tempo *t*

*(per esempio: il blocco 20 è assegnato alla pagina 5 del processo B, con bit di uso 0,e il puntatore è posizionato sul blocco 12)*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S. O. | S. O. | S. O. | S. O. | S. O. | S. O. |  |  |  |  |  |  | C  3  0 | C  5  0 | ∅ | D  1  0 |  |  | A  0  0 |  | ∅ | B  1  0 | C  7  0 | C  9  0 | C  8  0 | E  0  0 | D  8  0 | ∅ |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |

Configurazione della *CoreMap* e posizione del puntatore al termine dell’intervento di *PageDaemon*

Pagine scaricate: (C, 0), (B, 5), (A, 9) Processi scaricati: ∅

**Ipotesi b)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S. O. | S. O. | S. O. | S. O. | S. O. | S. O. | A  2  1 | ∅ | B  2  1 | B  4  1 | A  5  0 | ∅ | C  3  1 | C  5  1 | C  0  0 | D  1  1 | ∅ | ∅ | A  0  1 | ∅ | B  5  0 | B  1  1 | ∅ | C  9  1 | C  8  1 | E  0  1 | D  8  1 | ∅ | ∅ | A  7  1 | E  1  0 | E  2  1 | ∅ | B  9  1 | C  6  0 | B  11  1 | C  4  0 | ∅ | B  12  1 | ∅ |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |

Configurazione della *CoreMap* e posizione del puntatore al tempo *t*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S. O. | S. O. | S. O. | S. O. | S. O. | S. O. |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |

Configurazione della *CoreMap* e posizione del puntatore al termine dell’intervento di *PageDaemon*

Pagine scaricate: ∅ Processi scaricati: ∅

**Ipotesi c)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S. O. | S. O. | S. O. | S. O. | S. O. | S. O. | A  2  1 | A  3  0 | B  2  1 | B  4  1 | A  5  0 | ∅ | C  3  1 | C  5  1 | C  0  0 | D  1  1 | D  3  1 | ∅ | A  0  1 | ∅ | B  5  0 | B  1  1 | C  7  1 | C  9  1 | C  8  1 | E  0  1 | D  8  1 | A  9  0 | ∅ | A  7  1 | E  1  0 | E  2  1 | D  5  1 | B  9  1 | C  6  0 | B  11  1 | C  4  0 | ∅ | B  12  1 | E  3  1 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |

Configurazione della *CoreMap* e posizione del puntatore al tempo *t*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S. O. | S. O. | S. O. | S. O. | S. O. | S. O. | A  2  0 | ∅ | B  2  0 | B  4  0 | ∅ |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | ∅ | E  2  0 | D  5  0 | B  9  0 | ∅ | B  11  0 | ∅ |  | B  12  0 | E  3  0 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |

Configurazione della *CoreMap* e posizione del puntatore al termine dell’intervento di *PageDaemon*

Pagine scaricate: (E, 1), (C, 6), (C, 4), (A, 3), (A, 5) Processi scaricati: ∅

**Ipotesi d)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S. O. | S. O. | S. O. | S. O. | S. O. | S. O. | A  2  1 | A  3  0 | B  2  1 | B  4  1 | A  5  0 | D  2  1 | C  3  1 | C  5  1 | C  0  0 | D  1  1 | D  3  0 | ∅ | A  0  1 | D  5  0 | B  5  0 | B  1  1 | C  7  1 | C  9  1 | C  8  1 | E  0  1 | D  8  1 | A  9  0 | E  8  1 | A  7  1 | E  1  0 | E  2  1 | E  9  0 | B  9  1 | C  6  0 | B  11  1 | C  4  0 | ∅ | B  12  1 | E  3  1 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |

Configurazione della *CoreMap* e posizione del puntatore al tempo *t*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S. O. | S. O. | S. O. | S. O. | S. O. | S. O. | ∅ | ∅ | ∅ | ∅ | ∅ |  |  |  |  |  |  |  | ∅ |  | ∅ | ∅ |  |  |  |  |  | ∅ |  | ∅ |  |  |  | ∅ |  | ∅ |  |  | ∅ |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |

Configurazione della *CoreMap* e posizione del puntatore al termine dell’intervento di *PageDaemon*

Pagine scaricate: ∅ Processi scaricati: A, B

**ESERCIZIO M-23 – Working Set Manager**

Si consideri un sistema che gestisce la memoria con paginazione a domanda, applicando un algoritmo di sostituzione LRU locale e una politica di controllo dinamico del Working Set. Per ogni processo sono definiti i seguenti dati:

* l’intero *MaxBlocchi:* massimo numero di blocchi disponibili per il caricamento del *Working Set,* che viene ridefinito periodicamente dal demone *WorkingSetManager;*
* l’intero *PagineResidenti,* uguale al numero di pagine attualmente caricate in memoria e variabile nel tempo;
* la tabella delle pagine di ogni processo, con campi *Pagina, Blocco, R (bit di pagina riferita) e DP (Distanza Passata).*

Quando un processo avanza, per ogni pagina riferita:

* se il riferimento determina *Page Fault*, la pagina riferita viene caricata nel blocco libero individuato dal primo elemento della lista *BlocchiDisponibili* (che su suppone sempre non vuota), viene incrementata la variabile *PagineResidenti* e viene definito *DP= 0;*
* in ogni caso, si assegna *R= 1.*

Il processo *WorkingSetManager,* che interviene periodicamente, esegue le seguenti operazioni per ogni processo:

Fase 1) per ogni pagina residente in memoria aggiorna la distanza passata con il seguente algoritmo:

* se *R= 0* assegna *DP= DP+ 1;*
* altrimenti assegna *R= 0* e *DP= 0*

Fase 2) se *PagineResidenti> MaxBlocchi:*

* scarica dalla memoria principale *PagineResidenti- MaxBlocchi*, selezionandole in ordine decrescente del valore di *DP;*
* se per l’ultima pagina scaricata si ha *DP< 7,* assegna *MaxBlocchi= MaxBlocchi + 1;*

se *PagineResidenti< MaxBlocchi,* assegna *MaxBlocchi= MaxBlocchi – 1.*

Al tempo T1, subito dopo un intervento di *WorkingSetManager*, per il processo *P* si ha ***MaxBlocchi* = 7** e la tabella delle pagine ha la seguente configurazione:

|  |  |  |  |
| --- | --- | --- | --- |
| Pagina | Blocco | R | DP |
| 0 | - | - | - |
| 1 | 6 | 0 | 2 |
| 2 | - | - | - |
| 3 | - | - | - |
| 4 | - | - | - |
| 5 | 8 | 0 | 5 |
| 6 | - | - | - |
| 7 | 10 | 0 | 4 |
| 8 | 15 | 0 | 8 |
| 9 | - | - | - |
| 10 | 20 | 0 | 9 |
| 11 | - | - | - |
| 12 | 30 | 0 | 7 |

Dopo *T1* e prima del tempo *T2* il processo *P* avanza e riferisce nell’ordine le seguenti pagine:  ***5, 3, 0, 5, 0, 9, 3, 8, 9, 11***

Al tempo *T2* entra in esecuzione *WorkingSetManager*, che applica la politica sopra definita.

Al tempo *T4* termina l’intervento di *WorkingSetManager*

Supponendo che il contenuto della lista *BlocchiDisponibili* al tempo T1 sia 🡪 50 🡪 51 🡪 52 🡪 53 🡪 54 🡪 …., si chiede:

1. il valore di  *PagineResidenti* e la configurazione della Tabella delle Pagine del processo P al tempo *T2*;
2. la configurazione della Tabella delle Pagine del processo *P* al tempo *T3*, quando termina la Fase 1 di *WorkingSetManager*;
3. il valore di *PagineResidenti* e di *MaxBlocchi* ela configurazione della Tabella delle Pagine del processo *P* al tempo *T4,* quando termina la Fase 2 di *WorkingSetManager*.

**SOLUZIONE**

Tempo T2: Tabella delle Pagine

|  |  |  |  |
| --- | --- | --- | --- |
| Pagina | Blocco | R | DP |
| 0 | 51 | 1 | - |
| 1 | 6 | 0 | 2 |
| 2 | - | - | - |
| 3 | 50 | 1 | - |
| 4 | - | - | - |
| 5 | 8 | 1 | 5 |
| 6 | - | - | - |
| 7 | 10 | 0 | 4 |
| 8 | 15 | 1 | 8 |
| 9 | 52 | 1 | - |
| 10 | 20 | 0 | 9 |
| 11 | 53 | 1 | - |
| 12 | 30 | 0 | 7 |

Pagine Residenti= 10

Tempo T3: Tabella delle Pagine

|  |  |  |  |
| --- | --- | --- | --- |
| Pagina | Blocco | R | DP |
| 0 | 51 | 0 | 0 |
| 1 | 6 | 0 | 3 |
| 2 | - | - | - |
| 3 | 50 | 0 | 0 |
| 4 | - | - | - |
| 5 | 8 | 0 | 0 |
| 6 | - | - | - |
| 7 | 10 | 0 | 5 |
| 8 | 15 | 0 | 0 |
| 9 | 52 | 0 | 0 |
| 10 | 20 | 0 | 10 |
| 11 | 53 | 0 | 0 |
| 12 | 30 | 0 | 8 |

Tempo T4: Tabella delle Pagine

|  |  |  |  |
| --- | --- | --- | --- |
| Pagina | Blocco | R | DP |
| 0 | 51 | 0 | 0 |
| 1 | 6 | 0 | 3 |
| 2 | - | - | - |
| 3 | 50 | 0 | 0 |
| 4 | - | - | - |
| 5 | 8 | 0 | 0 |
| 6 | - | - | - |
| 7 | - | - | - |
| 8 | 15 | 0 | 0 |
| 9 | 52 | 0 | 0 |
| 10 | - | - | - |
| 11 | 53 | 0 | 0 |
| 12 | - | - | - |

Pagine Residenti= 7; MaxBlocchi= 8